CS111 Jeopardy: The Home Version

The game that turns CS111 into CSfun11!

Spring 2018

This is intended to be a fun way to review some of the topics that will be on the CS111 final exam. These questions are not indicative of the style and difficulty of the questions that will be on the final.

QUESTIONS

---

List One-Liners

List One-Liners [1] This is an expression whose value is the number of elements in the list L.

List One-Liners [2] This is a one-line expression whose value is a list of all even numbers between 1 and 100 (inclusive).

List One-Liners [3] This is a one-line statement in the body of

```python
with open('essay.txt', 'r') as infile:
    # print the total number of words in the file.
```

List One-Liners [4] Given a list L of single-character digit strings, this is a one-line expression whose value is the integer that corresponds to concatenating the digits in reverse order. For example:

- if L is the list ['3', '4', '5'], the code should compute 543
- if L is the list ['5', '3', '7', '2'], the code should compute 2735

List One-Liners [5] Given a list L, this is a one-line statement that moves the last element of the list to the beginning of the list (shifting all other elements to the right). It doesn’t create a new list, it mutates the original.

---

Iteration & Recursion

Iteration & Recursion [1] This is the definition of a function that satisfies the following contract:

```python
def sumLengths(listOfStrings):
    """Returns the sum of the lengths of all the strings in the given list of strings."
    """
```
Iteration & Recursion [2] This is the definition of a recursive function that satisfies the following contract:

```python
def sumRange(start, end):
    """Returns the sum of all integers in the range between the 1st argument, inclusive, and the 2nd argument, exclusive. """
```

For example:

- `sumRange(5, 7)` returns 11
- `sumRange(-2, 4)` returns 3
- `sumRange(3,3)` returns 0
- `sumRange(4,3)` returns 0

Iteration & Recursion [3] Consider the following function.

```python
def mystery(x):
    print(x)
    if x < 5:
        mystery(3*x - 1)
    elif x > 5:
        mystery(x - 3)
```

This is printed by the invocation `mystery(10)`.

Iteration & Recursion [4] This is the definition of a recursive function that satisfies the following contract:

```python
def whichPowerOf2(n):
    """Returns the power for a given power of 2."""
```

For example:

- `whichPowerOf2(1)` returns 0 (because $2^0 = 1$)
- `whichPowerOf2(8)` returns 3 (because $2^3 = 8$)
- `whichPowerOf2(16)` returns 4 (because $2^4 = 16$)

Iteration & Recursion [5] These statements read the content of tweets retrieved from the Twitter API, and print this list of hashtags: `["ruhlman18", "redClass18", "wellesley",...]. (The list may contain duplicates.)

```python
tweets = [
    {
        "source": "Come to PNE 249 for our Ruhlman! #ruhlman18", "entities": {
            "urls": [], "hashtags": ["ruhlman18"]
        },
        "source": "Congratulations to our seniors! #purpleClass18, #wellesley", "entities": {
            "urls": [], "hashtags": ["purpleClass18", "wellesley"]
        }
    }
]
```

Dictionaries & Sets

Dictionaries & Sets [1] This is the value of the expression `len(set('abracadabra'))`
Dictionaries & Sets [2] These are all the following types that cannot be keys in a dictionary:

- int
- bool
- string
- list
- tuple
- set
- dict

Dictionaries & Sets [3] These are all of the following boolean variables that are not necessarily True:

\[
\begin{align*}
\text{bool1} &= \{'x':1, 'y':2, 'z':3\} == \text{dict}([('x', 1), ('y', 2), ('z', 3)]) \\
\text{bool2} &= \{'x':1, 'y':2, 'z':3\}.\text{items()} == [(x', 1), ('y', 2), ('z', 3)] \\
\text{bool3} &= \text{set}({'x':1, 'y':2, 'z':3}.\text{items()}) == \{(x', 1), ('y', 2), ('z', 3)}
\end{align*}
\]

Dictionaries & Sets [4] This one variable has a value that is not equal to the others:

\[
\begin{align*}
d1 &= \{'a':3, 'b':2, 'c':1\} \\
d2 &= \text{dict}([('b', 2), ('c', 1), ('a', 3)]) \\
d3 &= \{\text{key}: 'bacaba'.\text{count(key)} \text{ for key in 'cab'}\} \\
d4 &= \text{dict}(.\text{zip('cba', [1, 2, 3]))} \\
d5 &= \text{json.loads('"b":2, "a":3, "c":1')} \\
d6 &= \{(a', 3), ('b', 2), ('c', 1)}
\end{align*}
\]

Dictionaries & Sets [5] This is a printed representation of the value returned by \textit{mysteryDict('a bat ate an oval berry')} given the following definition:

\[
\begin{align*}
def \text{mysteryDict(sentence)}: \\
\text{dct} &= \{} \\
\text{for word in sentence.split():} \\
\text{c} &= \text{word[0]} \\
\text{dct[c]} &= \text{dct.get(c, 0) + len(word)} \\
\text{return dct}
\end{align*}
\]

Objects

Objects [1] A class declaration typically includes these entities, used to keep track of an object’s state.

Objects [2] Consider the following classes:

\[
\begin{align*}
\text{class Food}: \\
\text{def info(self):} \\
\text{print('Good to eat')}
\end{align*}
\]

\[
\begin{align*}
\text{class Dessert(Food):} \\
\text{def calories(self):} \\
\text{print('Lots of calories')}
\end{align*}
\]

\[
\begin{align*}
\text{class Cake(Dessert):} \\
\text{def flavor(self):} \\
\text{print('I like chocolate')}
\end{align*}
\]

This is the number of user-defined methods that a \textit{Cake} object has.
Objects [3] This will be printed by the following program.

class A():
    def number(self):
        print(8)

class B(A):
    def number(self):
        A.number(self)
        print(9)

class C(B):
    def number(self):
        print(7)
        B.number(self)

C().number() # C() invokes the default zero-argument constructor for a class

Objects [4] This will be printed by the following program.

import math
class RightTriangle:
    def __init__(self, base, height):
        self.base, self.height = base, height

    def scale(self, factor):
        self.base, self.height = self.base*factor, self.height*factor

    def hypotenuse(self):
        return math.sqrt(self.base**2 + self.height**2)

    def perimeter(self):
        return self.base + self.height + self.hypotenuse()

tri = RightTriangle(6, 8)
tri.scale(0.5)
print tri.perimeter()

Objects [5] These statements in a MinionYoga step method make its instances rotate by 90 degrees half the time:

import random

class MinionCeiling(Minion):
    def step(self):
        # most of body omitted for space reasons
        else: self.minionLayer.move(self.deltax, -self.deltay)

class MinionYoga(MinionCeiling):
    '''Rotate the minion by 90 degrees 50% of the time'''
    def step(self):
        # FILL IN THE MISSING STATEMENTS
Bugs That Bite

Bugs That Bite [1] This is a bug in the following function definition:

```python
def compare(a, b):
    if a == b:
        return 'equal'
    else:
        return 'not equal'
```

Bugs That Bite [2] This is a bug in the following class definition:

```python
class Animal:
    def __init__(self, numLegs):
        numberOfLegs = numLegs

    def isBiped(self):
        return self.numberOfLegs == 2
```

Bugs That Bite [3] Recall that `random.randint(a, b)` returns a random integer i such that \(a \leq i \leq b\). This is a bug in the following function definition:

```python
import random
def chooseRandom(aList):
    if len(aList) > 0:
        randomIndex = random.randint(0, len(aList))
        return aList[randomIndex]
```

Bugs That Bite [4] This is a bug in the following code:

```python
vowelDict = {}
for vowel in 'aeiou':
    vowelDict[vowel] = vowel.upper()
print(vowelDict['E'])
```

Bugs That Bite [5] The following definition of the `areAllPositive` function does not satisfy the contract specified in its comment. Show this by giving a sample input on which it returns an incorrect answer.

```python
def areAllPositive(numbers):
    '''Returns True if all elements in the list of numbers are positive, and False otherwise.'''
    if len(numbers) == 0:
        return True # *correct* answer for so-called ``vacuously true` case
    else:
        for num in numbers:
            if num <= 0:
                return False
        else:
            return True
When the below program is executed, this will be printed.

```python
x = 3
y = 8
def f():
    x = 6
    y = 7
f()
print(x)
print(y)
```

This is the definition of a function `swap` that takes three arguments (a list `L` and two list indices `i` and `j`) and modifies `L` by swapping the contents of its slots at indices `i` and `j`.

Consider the function below:

```python
def appendages(L):
    if len(L)==0:
        return []
    else:
        return L + appendages(L[1:])
```

This is the list returned by the invocation `appendages([1,2,3,4])`.

This is (1) the buggy expression and (2) the corrected expression in the following function definition:

```python
def frequenciesBuggy(strings):
    """Returns a dictionary mapping each string in the given list of strings to the number of times it appears in the list."
    freqDict = {}
    for s in strings:
        freqDict[s] = 1 + freqDict[s]
    return freqDict
```

This is a function that satisfies the following contract:

```python
def countOfMostCommonCharacter(s):
    """Returns the number of times the most commonly occurring character in the string `s` occurs."
    For example:
    • `countOfMostCommonCharacter('eerie')` returns 3
    • `countOfMostCommonCharacter('Mississippi')` returns 4
ANSWERS

List One-Liners

List One-Liners [1]  len(L)

List One-Liners [2]
Here are some of many possible answers:

- [i for i in range(1, 101) if i\%2==0]
- filter(lambda i: i\%2==0, range(1,101))
- range(2,101)[::2]
- range(2,101,2)

List One-Liners [3]  len(infile.read().split())

List One-Liners [4]  int(''.join(L)[::-1])

List One-Liners [5]  L.insert(0, L.pop())

Iteration & Recursion

Iteration & Recursion [1]
Here are some of many solutions:

Solution 1:
```python
def sumLengths(listOfStrings):
    sum = 0
    for s in listOfStrings:
        sum += len(s)
    return sum
```

Solution 2:
```python
def sumLengths(listOfStrings):
    return sum([len(s) for s in listOfStrings])
```

Solution 3:
```python
def sumLengths(listOfStrings):
    return sum(map(len, listOfStrings))
```

Iteration & Recursion [2]
```python
def sumRange(start, end):
    if start>=end:
        return 0
    else:
        return start + sumRange(start+1, end)
```
Iteration & Recursion [3]
10
7
4
11
8
5

Iteration & Recursion [4]
def whichPowerOf2(n):
    if n==1:
        return 0
    else:
        return 1 + whichPowerOf2(n/2)

Iteration & Recursion [5]
Solution 1:
hashtags = []
for tweet in tweets:
    hashtags += tweet['entities']['hashtags']
print hashtags

Solution 2:
hashtags = []
for tweet in tweets:
    hashtags.extend(tweet['entities']['hashtags'])
print hashtags

Dictionaries & Sets

Dictionaries & Sets [1] 5. set('abracadabra') is {'a', 'b', 'c', 'd', 'r'}, which has 5 elements.

Dictionaries & Sets [2] list, set, and dict These are mutable object types that cannot be dictionary keys.

Dictionaries & Sets [3] bool2. Although bool2 could be True, it can also be False, because the ordering of pairs returned by .items() is unpredictable. bool1 and bool3 are necessarily true.

Dictionaries & Sets [4] d6. It is a set of tuples, not a dictionary. But dict(d6) would be dictionary equal to the others.

Dictionaries & Sets [5] {'a': 6, "b": 8, "o": 4}. The order of key/value pairs is arbitrary, as is using double or single quotes for the string keys.

Objects

Objects [1] instance variables, state variables, or data attributes. (instance variables is standard across object-oriented programming languages; state variables is a more generic term that means the same thing in an object-oriented context. The term data attributes is specific to Python.)

Objects [2] 3 methods (info, calories, flavor). This does not include default object methods like __repr__, and does not include __init__ (which is used to create the instance, but not operate on it after it has been created).
Joke: Why was 6 afraid of 7? Because 7 8 9!

**Objects [4]** 12.0 The hypotenuse of 3.0,4.0 triangle is 5.0, and $3.0 + 4.0 + 5.0 = 12.0$. The result is necessarily a float, not an integer, both because of the multiplication by 0.5 (which returns a float) and the use of `math.sqrt` (which always returns a float).

**Objects [5]**
```python
def step(self):
    MinionCeiling.step(self)  # move up toward ceiling
    if random.randint(0,1) == 0:
        self.minionLayer.rotate(90)
```

---

**Bugs That Bite**

**Bugs That Bite [1]**  $a = b$ should be $a == b$

**Bugs That Bite [2]** In the `__init__` method, `numberOfLegs = numLegs` assigns to the local variable `numberOfLegs` in the execution frame for the `__init__` method, but does not create an instance variable in the new `Animal` instance. This can be fixed by changing this line to `self.numberOfLegs = numLegs`.

**Bugs That Bite [3]** `random.randint(0, len(aList))` is inclusive on its second argument. So in the case where `randomIndex` is `len(aList)`, the error `list index out of range` will be raised. The correct expression is `random.randint(0, len(aList)-1)`.

**Bugs That Bite [4]** `vowelDict['E']` raises the error `Key Error 'E'` because 'E' is not a key in `vowelDict` (but 'e' is).

**Bugs That Bite [5]** `areAllPositive` is wrong because its return value is based only on the first element of the list. Any list with a positive first element and some later nonpositive element will be incorrect. For example, `brokenAreAllPositive([3,-2])` returns `True`.

---

**Potpourri**

**Potpourri [1]** The assignments to `x` and `y` in the body of the function `f` create local variables in the execution frame for `f` and do not change the values of the global variables `x` and `y`. So the answer is:

```python
3
8
```

**Potpourri [2]**

Solution 1:
```python
def swap(L, i, j):
    ival = L[i]
    L[i] = L[j]
    L[j] = ival
```
Solution 2:

def swap(L, i, j):
    ival, jval = L[i], L[j]  # simultaneous assignment
    L[i] = jval
    L[j] = ival

Solution 3:

def swap(L, i, j):
    L[j], L[i] = L[i], L[j]  # simultaneous assignment to list slots!

Potpourri [3] [1, 2, 3, 4, 2, 3, 4, 3, 4, 4]

Potpourri [4]

1. \(1 + \text{freqDict}[s]\) (because when the string \(s\) is not yet in \text{freqDict}, there is a key error)

2. \(1 + \text{freqDict}.get(s, 0)\) (because this evaluates to 1 when \(s\) is not in \text{freqDict})

Potpourri [5]

Solution 1:

def countOfMostCommonCharacter(s):
    countDict = {}
    for ch in s:
        if ch not in countDict:
            countDict[ch] = 1
        else:
            countDict[ch] += 1
    return max(countDict.values())

Solution 2:

def countOfMostCommonCharacter(s):
    countDict = {}
    for ch in s:
        countDict[ch] = countDict.get(ch, 0) + 1
    return max(countDict.values())

Solution 3:

def countOfMostCommonCharacter(s):
    maxCount = 0
    for ch in s:
        maxCount = max(maxCount, s.count(ch))
    return maxCount

Solution 4:

def countOfMostCommonCharacter(s):
    return max([s.count(ch) for ch in s])

Solution 5:

def countOfMostCommonCharacter(s):
    return max(map(lambda ch: s.count(ch), s))